BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **15**, No 3

Sofia • 2015

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2015-0048

Performance Guarantee Mechanism for Multi-Tenancy SaaS Service Based on Kalman Filtering

Liu Yanpei[1, 2], *Li Chunlin*[1] ,*Yang Zhiyong*[1] , *Chen Yuxuan*[3] , *Xu Lijun*[1]

[1] *Department of Computer Science, Wuhan University of Technology, Wuhan, China*
[2] *Information Engineering College, Henan Institute of Science and Technology, Xinxiang, China*
[3] *Department of Microelectronics and Solid-State Electronics, University of Electronics Science and Technology of China, Chengdu, China*
Email: liupeiok123@gmail.com

Abstract: *This paper proposes a special System Architecture for Multi-tenancy SaaS Service* (*SAMSS*), *which studies the performance security issues at the business logic layer and data processing layer respectively. The Kalman filtering Admission Control algorithm* (*KAC*) *and the Greedy Copy Management algorithm* (*GCM*) *are proposed. At the business logic layer, Kalman filtering admission control algorithm is presented. It uses a Kalman filter to conduct the dynamic evaluation for the CPU resource for multi-tenancy SaaS service and reduces the unnecessary performance expenses caused by direct measurement of CPU resources. At the data processing layer, the Greedy Copy Management algorithm* (*GCM*) *is presented. It changes the copy placement as a K-partitioning set partitioning problem and adopts a greedy strategy to reduce the number of times for creating a data copy. Finally, the experimental analysis and results prove the feasibility and efficiency of the algorithms proposed.*

Keywords: *Kalman filtering, business logic layer, data processing layer, multi-tenancy, SaaS.*

1. Introduction

Software as a Service (SaaS) is a way of software deployment. Tenants of the order services have authorized its end users to access the software through the network on demand, where the end users share the applications and data. Multi-tenancy enables the concurrent users from different tenants to share the same infrastructure

resources, for the sake of cost reducing and income increasing, which is one of the key features of SaaS service under the circumstances of large-scale cloud computing [1-3].

Performance guarantee for multi-tenancy SaaS service has been focused on in recent studies. Flash crowd [4] is a large number of tenants requesting SaaS service simultaneously, causing the server to overload and the server buffer to be completely consumed. Large numbers of request packets have been discarded. Due to the computing resource limits, the accepted tenant requests must wait long for the allocated computing resources, that causes a great impact on the delay performance, the tenants tend to give up the service because of a packet loss or too long waiting time, resulting in great loss. In addition, with the rapid increase of SaaS service tenants and the random appearance of tenants' Flash crowd, the data resources are facing great load and stress, which means that the data resources will become the bottleneck of the whole SaaS service. You need to adapt to the load fluctuation by dynamic changing the number of servers.The strategy of admission control and copy management is the key to the system performance guarantee mechanism for multi-tenancy SaaS service. As for the admission control strategy, García D F et al. [5] point out the requirements that should be met when the QoS control mechanism of service provider server works in B2B environment, putting forward the QoS control mechanism with a monitor and a controller. The service quality control algorithm makes schedules according to the user's priority. The QoS control mechanism ensures the system performance under circumstances of overload. Z h a n g et al. [6] propose a resource consumption estimate model in a multi-tenancy application environment, based on the maximum size of tenants with given nodes, figured out by the heuristic algorithm. But the present researches can hardly assess the time-varying resource state of SaaS services efficiently and the precision of the method depends on long time, high quality samples as inputs, which is easily affected by outliers caused by resource competition, thus generating errors in the admission control mechanism [7]. In addition, the current copy management strategy mostly adopts the method of creating copy numbers and fixed timing [8], so it is difficult to adapt to SaaS service by the dynamic change of the number of tenants, such as creating too many copies, for there will be waste of storage space. On the other hand, the system performance cannot be improved efficiently. In addition, a frequent migration of the copy data will consume the network bandwidth, thus reducing the performance of the system.

In the viewpoint of the problems above mentioned, this paper studies its performance guarantee respectively at the layer of business logic and data processing. At the business logic layer, Kalman filtering [9] was used to reflect the resource usage and surplus situation on different servers in time for CPU resources dynamic assessment for multi-tenancy SaaS service. It provides the basis for tenant's admission control mechanism. KAC, the Kalman filtering admission control algorithm was proposed. At the data processing layer, the system architecture is adapted to the dynamic changes of the load by dynamically adjusting the load distribution among every copy and the placement of a copy between the nodes. The optimal weighted rotation scheduling algorithm was used to realize the

distribution of the load. Also, the system architecture classified the copy placement into a K-partitioning [10] set partition problem and utilizes the greedy strategy to reduce the creation times. GCM, the algorithm of greedy copy management was proposed.

This paper is divided into five sections. Section 1 outlines the system architecture SAMSS for multi-tenancy SaaS service; Section 2 discusses the performance guarantee mechanism at the business logic layer; Section 3 describes the performance guarantee mechanism of the data management layer; Section 4 presents an experimental analysis for the performance guarantee technology. Finally, there is a summary of the paper.

2. System architecture for multi-tenancy SaaS service

The System Architecture for Multi-tenancy SaaS Service (SAMSS) discussed in this paper is made up of the business logic layer and data processing layer. A full service request process starts from the business logic layer, and it may need access to the data processing layer many times during the process until it generates a complete response and returns the response to the tenant.

The business logic layer is mainly intended to provide a certain degree of performance guarantee for tenants. The business logic layer of SAMSS mainly consists of resource dynamic evaluation components and an access control unit. The resource dynamic assessment components mainly use a Kalman filter to make resource consumption calculation for the multi-tenant SaaS service, so that the system resource usage and the remaining ones can be obtained timely to provide a basis for judgment at the next step of access control. The access control unit manages the access request of the tenants in the whole system to prevent the system from being overloaded and decides whether to accept the new tenant requests according to the usage of the server resource.

The data processing layer is mainly intended to provide high availability of data protection to tenants. SAMSS maintains multiple copies of data distributed on different nodes for each tenant. In SAMSS the reliability and access efficiency of each tenant’s data can be improved through the use of multiple nodes, so a request dispatcher used for load balancing between nodes is supposed to be added.

2.1. The performance guarantee mechanism of the business logic layer

This strategy uses a Kalman filter to do the dynamic assessment of CPU resources for multi-tenant SaaS services and reduces the direct measurement of CPU resources, when it needs to inject a probe and causes unnecessary performance overhead. Besides, the multi-tenant strategy access control is designed by using the method of resource allocation and resource reservation and it avoids the system’s overload caused by flash crowd tenants.

2.1.1. The dynamic evaluation of multi-tenant CPU resources based on a Kalman filter

Kalman filter has been widely used and studied in the fields of automatic control and auxiliary navigation, and its main characteristic is that it can use a form of

approximately optimal estimation based on an observable value to estimate unobservable value, and can update the former observed value as the new observed value comes out, therefore it is more suitable for online assessment of the time-varying resource state.

Kalman filter provides a general method to estimate the unobservable value x in discrete time points. The state x_k at point k can be defined as a linear stochastic difference equation:

(1) $$x_k = Ax_{k-1} + w_{k-1}.$$

The test value z_k at point k is defined as

(2) $$z_k = H_k x_k + v_k ,$$

where: A is the state transition matrix from the $k-1$ point to the k-th point; w_{k-1} is the process error; Q_{k-1} is the covariance matrix; H_k is the transition matrix from x_k to z_k; v_k is the observation error; R_k is the covariance matrix.

Multi-tenant SaaS service request process may involve a variety of resources. Those with higher resource utilization rate are called bottleneck resources. For different types of services, the bottleneck resources may be different, but the bottleneck resources for multi-tenant SaaS service are most likely to be the CPU resources. Therefore, in this paper we mainly consider the dynamic assessment of the CPU resources. For multi-tenant SaaS services, the prerequisite condition for dynamic assessment is collecting online server log information, including the tenant's throughput amount and CPU utilization rate of the server. During the service, runtime information is monitored and recorded at a fixed time interval, this interval is called a monitoring window. For convenience of discussion, to a server which is the host of N tenants, we give the following symbols and their meanings:

T indicates the size of the monitoring window;

N_i indicates the number of transactions that the i-th $(1 \le i \le N)$ tenant completes in the monitoring window;

U_{CPU} indicates the average CPU utilization rate of the server in the monitoring window;

S_i indicates the average service time of all the i-th tenant's transactions (namely, average CPU time occupied by all transactions).

According to the Utilization Law [11], the resource utilization rate is equal to the throughput amount multiplied by the service time, the equation obtained is as follows:

(3) $$U_{\mathrm{CPU}} T = \sum_i N_i S_i .$$

Due to the difficulty to measure the service time S_i accurately, we use C_i to represent its approximation, so as to obtain the calculation equation of the resource utilization U'_{CPU} rate approximation:

$$U'_{\mathrm{CPU}} = \frac{\sum_i N_i C_i}{T}. \tag{4}$$

A statistical analysis method can be used to solve C_i, for such indirect approximation solution of the problem, the error of U_{CPU} and U'_{CPU} is one of the typical indicators for measuring accuracy. Next, our goal is to work out how to reduce the error of the real service time S_i and the approximate service time C_i.

Firstly, we model the unobservable state *x* into an *N*-dimensional vector $x_k = (C_1^k, C_2^k, \cdots, C_N^k)$ which contains N tenants' average service time of transactions, and it indicates each tenant's average service time of transactions at point k. Then, according to Formula (3), we model the observed total CPU utilization rate z_k and we obtain:

$$z_k = \frac{\sum_i N_i^k C_i^k}{T} + v_k, \tag{5}$$

where N_i^k indicates the monitored throughput rate of tenant i; the transition matrix H_k from x_k to z_k is defined as $(\frac{N_1^k}{T}, \frac{N_2^k}{T}, \cdots, \frac{N_N^k}{T})$.

The Kalman filter algorithm makes iteration assessment on the service time at the end of each monitoring window, the initial value including the state initial value $\hat{x}_0$ and the initial error covariance matrix. The iteration process is as follows:

① forward projection of the state of x –

$$\hat{x}_k^- = A\hat{x}_{k-1}; \tag{6}$$

② calculate forwards the covariance matrix of the state priori estimate error P_k^- –

$$P_k^- = AP_{k-1}A^{\mathrm{T}} + Q_k; \tag{7}$$

③ calculate the Kalman gain K_k –

$$K_k = P_k^- H_k^{\mathrm{T}} (H_k P_k^- H_k^{\mathrm{T}} + R_k)^{-1}; \tag{8}$$

④ update the state of x according to the observed variable z_k –

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k\hat{x}_k^-); \tag{9}$$

⑤ calculate the covariance matrix P_k of the estimate error after the state has been updated –

$$P_k = (I - K_k H_k)P_k^-. \tag{10}$$

In the process of iteration, step ④ which modifies the state of x is the key to update the estimated value, this equation can be simplified in the form of $x_{\mathrm{new}} = x_{\mathrm{old}} + K \bullet e$, which is to say, that *K* can be regarded as the weight matrix of the

modified x, and the error e and the corresponding weight are used to correct the data x_{old} At the same time, in the calculation of step ④, it is also necessary to consider to set the valuation range of the average service time for each tenant affair, namely non-negative and less than a certain upper bound: $0 \le C_i^k < u_k$, where u_k is the upper bound of the estimated value. In this paper we set the range of state x as: $(0, (\mu u_k + (1-\mu) x_{k-1}))$, where $\mu = 0.9$ and $u_k = U^k_{\text{CPU}} T \,/\, N_i^k$. Then the calculation at step ④ is revised as:

(11) $$\hat{x}_k^{\text{trial}} = \hat{x}_k^- + K_k (z_k - H_k \hat{x}_k^-),$$

(12) $$\hat{x} = \min(\hat{x}_k^{\text{trial}}, (\mu u_k + (1-\mu) x_{k-1})).$$

When there is any tenant requesting to be allowed to enter the system, the system calculates the resource consumption by using the multi-tenancy CPU resource dynamic assessment method based on Kalman filtering. The usage and the rest of the system resource can be known in time, providing the basis for the next tenant's admission control mechanism, so that that the delay, due to making a decision by monitoring the response time, can be avoided, reducing the unnecessary performance expenses caused by direct measurement of CPU resources which needs an injected probe.

2.1.2. Admission control mechanism

In the proposed system framework for multi-tenancy SaaS service, there are three dependent services represented, respectively standing for tenant's three kinds of services, where there is a competition for service S3 between (S1, S2, S3) and (S4, S5, S3), which is very common in the multi-tenancy SaaS service system.

To illustrate our control strategy, we deal with the n requests from tenant t, and only considering the CPU resource as an example. When a tenant request arrives, the system will begin to implement the admission control algorithm, which includes the allocation part and the reserve part of resources. In the allocation part, if the remaining resources in the server where there are service instances of S1, S2, S3 are greater than the specified minimum remaining resources of each server, the algorithm will firstly work out the acceptable request number n. Once a new request is accepted, the system will recalculate the remaining resources of R_{S1}, R_{S2}, R_{S3} by using a multi-tenancy CPU resource dynamic assessment method based on Kalman filtering, and at the same time, it will record its online tenants numbers according to the request category. If the remaining resources in the servers of S1, S2, S3 are less than the minimum remaining resources, this means it has attained a critical load state. If the tenants are still a lot, we can only accept a limited number of accessed requests from a part of the senior tenants (tenant classification: senior tenants, intermediate tenants and primary tenants). This efficiently improves the impact the system has on the performance of senior tenants with heavy load. In order to further improve the system's ability to handle the request, we set a buffer queue in the controller, to store temporarily in the cache queue a certain number of access requests rejected due to overload protection. Once the system resources have a rest, these requests will be processed in time. We call it the Kalman filtering Admission Control algorithm (KAC).

2.2. The performance guarantee mechanism of the data processing layer

With the rapid increase of SaaS service tenants and the random appearance of tenants' Flash crowd, the data resources are under great load and pressure, which means that the data resources will become the bottleneck of the whole SaaS service; hence a performance guarantee mechanism is required for the layer of data processing. Within the data processing layer, SAMSS will allocate each data copy to each node and use the data-copying algorithm according to [12] to ensure the consistency of each copy. The point of this section is to seek out how to adjust the allocation of loads among the copies and the placement of copies among nodes to guarantee the performance indicators at the data processing layer on the basis of ensuring the consistency of each copy. At present, the setting of the load distribution strategy [13, 14] and the copy placement strategy [15-17] are facing the following problems.

Setting the load distribution strategy, under the premise of fixed copy placement, how to allocate the read only load to each node has become a key sub-problem for the performance safeguard mechanism. The difficulty of this sub-problem is how to improve the overall performance of the system under the restriction of the state by balancing the load of each node as much as possible. The setting of a copy placement strategy, the dramatic change of the load makes the work of balancing each node only by load distribution difficult, hence the system must adjust the mapping of copies to nodes and the gross of data nodes to have the load balanced. In this process, the system must minimize the usage of resources and the moves of copies, which increase the difficulty of setting of the copy placement strategy.

This section focuses mainly on studying the above two problems. In order to facilitate the discussion, we list the following symbols and their meanings:

S – the set of data nodes, j of which is denoted by s_j, and $\forall j \in [1, M]$;

T – the tenants collection, i of which is denoted by t_i, and $\forall i \in [1, N]$;

L – the system's overall load vector, namely $L =< L_1, L_2, \cdots, L_N >$;

$\mathrm{SF}_{i,j}$ – the Stretch Factor indicator of the request of the tenant t_i when performed on s_j;

R_i^{T} – the copy number of tenant t_i, and $R_{\max}^{\mathrm{T}}, R_{\min}^{\mathrm{T}}, R_{\mathrm{default}}^{\mathrm{T}}$, respectively represent the maximum, the minimum and the default value of R_i^{T};

K – the vector the server uses; when $k_j = 1$, it means that the server is in a running state, and when $k_j = 0$ just the opposite;

A – the placed matrix of a copy of the data, namely

$$A=\begin{bmatrix} A_{1,1} & \cdots & A_{1,M} \\ \vdots & \ddots & \vdots \\ A_{N,1} & \cdots & A_{N,M} \end{bmatrix};$$

if server s_j maintains a copy of the data of tenant t_i then $A_{i,j}=1$, otherwise $A_{i,j}=0$;

A_0 – the generated placed matrix of copy in the process of the last copy's adjustment;

λ – the load distribution matrix, namely

$$\lambda=\begin{bmatrix} \lambda_{1,1} & \cdots & \lambda_{1,M} \\ \vdots & \ddots & \vdots \\ \lambda_{N,1} & \cdots & \lambda_{N,M} \end{bmatrix},$$

where $\lambda_{i,j}$ represents the request rate when the load of tenant t_i is allocated to the server node s_j, furthermore, $\lambda_{i,j}^{R}$ represents the request rate when the read load of tenant t_i is allocated to the server node s_j, and $\lambda_{i,j}^{W}$ represents the request rate when the load of tenant t_i is allocated to the server node s_j.

SF_{max} – the elongation factor threshold of the data query request;

U_{max} – the biggest resource utilization threshold of each node;

U_{target} – the overall goal resource utilization of the system;

$U_{upperThreshold}$ – the floating range of the upper limit of the overall resource utilization of the system;

$U_{lowerThreshold}$ – the floating range of the lower limit of the overall resource utilization of the system.

2.2.1. Load distribution

In order to accurately analyze the load distribution problems, it is necessary to define the metrics of the load balancing degree, and establish the performance model of data nodes, which is used to estimate the effect of adjustment before setting the real resource management strategies

Taking lessons from previous works, this paper uses a queue having multiple types of M/G/1/ PS requests as the performance model of each data node.

According to the queuing theory results, the requesting elongation factor performed on node s_j is

$$\mathrm{SF}_{i,j} = \frac{1}{1-U_j} = \frac{1}{1-\sum_{i=1}^{N}(\lambda_{i,j}^{W} D_i^{W} + \lambda_{i,j}^{R} D_i^{R})}, \tag{13}$$

where node U_j represents the resource utilization rate, and $0 \le U_j < 1$; D_i^W and D_i^R respectively represent the write data rate and read data rate of tenant t_i.

This paper takes the load inclination rate as the cluster node load imbalance degree of metrics, the formal definition is as follows:

$$\mathrm{Slope} = \sum_{j=1}^{M} \frac{1}{1-U_j}. \tag{14}$$

Thus the features of the load inclination rate could be concluded:

1. When a node's resource utilization rate is high, its contribution to the load inclination rate will greatly improve. When a node's resource utilization rate approaches 100%, the system load inclination rate approaches infinity.

2. When each data node is at the same resource utilization rate, the load inclination will be minimum.

Based on the above performance model and the metrics, we could further form the load balancing problem into the following formula:

$$\min \mathrm{Slope}(\lambda) = \sum_{j=1}^{M} \frac{1}{1-\sum_{i=1}^{N}\left(\lambda_{i,j}^{W} D_i^{W} + \lambda_{i,j}^{R} D_i^{R}\right)}, \tag{15}$$

where the reasonable value range of read requests distribution is $\forall i \in [1,N], j \in [1,M], \sum_{j=1}^{M} \lambda_{i,j}^{R} = L_i^R$, if $A_{i,j} = 0$ then $\lambda_{i,j}^{R} = 0$, $0 \le \lambda_{i,j}^{R} \le L_i^R$; the reasonable value range of the write requests distribution is $\lambda_{i,j}^{W} = \begin{cases} L_i^W, A_{i,j} = 1 \\ 0, A_{i,j} = 0 \end{cases}$, which also requires that the load summation of each node distributed must be less than its processing power, namely $\sum_{i=1}^{N} [\lambda_{i,j}^{W} D_i^{W} + \lambda_{i,j}^{R} D_i^{R}] < 1$.

The above issues belong to a special kind of the load distribution problem; the ideal state is the system to be able to get the load information of each node in a data copy in the distribution of each tenant request. For each tenant t_i request, we distribute it to the node which has the smallest queue length in the node collection of the data copy where t_i is stored. But it is the proper state to be able to get the data and the load information of each node in the copy in time; we call this algorithm the Shortest Queue First (SQF). In an actual situation, when each tenant request is distributed in the system, the load information of each node can only be obtained periodically. Then the Weighted Round-Robin scheduling algorithm is adopted. We use the loading intensity that each tenant t_i is assigned to the node s_j

as the weight $\lambda_{i,j}$ of the Weighted Round-Robin scheduling algorithm. This algorithm is called the Optimal Weighted Round-Robin (OWRR). The load distribution matrix obtained from this algorithm is called the Optimal Balanced Load (OBL).

2.2.2. The copy placement

When the load fluctuation is relatively severe, simply changing the distribution of load will make it difficult to accomplish the whole goal of this section that is minimizing the consumption of resources on condition that each performance index of the tenant is assured. To further implement the target, the dynamic adjustment to the copy placement is needed. Specifically, two kinds of performance management methods of the load distribution and the copy placement need to be considered simultaneously. The performance management goal on the data processing layer can be formally described as follows:

(16) $\min f(\lambda, A) = \sum_{j=1}^{M} k_j \quad \min g(\lambda, A) = | A - A_0 | \quad A_{i,j} \in \{0,1\}$,

where the number of data copy of each tenant has the following range $\forall i \in [1, N]$, $\sum_{j=1}^{M} A_{i,j} \in [R_{\min}^{\mathrm{T}}, R_{\max}^{\mathrm{T}}]$; and it is required that only nodes in the moving state can place the data copy, namely $\forall j \in [1, M]$, $k = \begin{cases} 1 \text{ if } \sum_{i=1}^{N} A_{i,j} > 0 \\ 0 \text{ if } \sum_{i=1}^{N} A_{i,j} = 0 \end{cases}$; the reasonable value range of read requests is $\lambda_{i,j}^{W} = \begin{cases} L_i^W, A_{i,j} = 1 \\ 0, A_{i,j} = 0 \end{cases}$; and the performance index for each tenant is bounded, namely $\frac{1}{1 - \sum_{i=1}^{N} (\lambda_{i,j}^{W} D_i^{W} + \lambda_{i,j}^{R} D_i^{R})} \le \mathrm{SF}_{\max}$.

Through classifying the problem above mentioned into a K-partitioning set partition problem, we can draw the conclusion that it is a NP-hard problem, so it is difficult to get the optimal solution. This paper uses a greedy strategy to try to find an optimal solution. We call it the GCM algorithm. GCM is composed of the node providing algorithm and the copy placement algorithm.

The node providing algorithm adapts to changes in the overall intensity of load by dynamically adjusting the number of nodes in the system. Specifically, when shrinking the number of nodes, the algorithm adopts the strategy of deleting the node with the lowest load first under the Optimal Balanced Load (OBL) scheme. Since deleting any nodes may make the amount of certain tenant data copy less than the node $R_{\min}^{\mathrm{T}}$, it is reasonable to move it to some reserved nodes rather than simply throw it away. Also, it is necessary to move the copy to the node with the lowest load first under the OBL scheme when a copy movement is required.

In essence, the copy placement algorithm is a kind of a greedy algorithm. Every time when the copy that needs adjustment is selected, the copy placement algorithm first chooses to completely reduce the extent of the copy that has an unbalanced load. The time complexity of this algorithm is $O(n)$, where n represents the number of nodes. The copy placement algorithm is used for the adjustment of the copy placement to adapt to the change of the proportion of each tenant in the system when the load varies. First of all, the copy placement algorithm calls the node providing algorithm to adjust the node volume. Subsequently, it keeps an eye on the load balance between nodes.

3. Experimental analysis

This section discussed the effect of the multi-tenant SaaS service performance guarantee mechanism separately on the business logic layer and data processing layer.

3.1. The business logic layer

The experimental environment is a typical three-tier architecture system, including the Tomcat server which deploys the services S1, S2, S3, S4 and a number of database servers. S1, S2, S3, S4 are deployed on the same server, the Apache JMeter 2.4 is run to simulate a tenant's load, and the average response time, throughput, and the other data is obtained through "Graph Result" listeners. All systems are run on Windows Server 2003. In order to verify the effect of the performance guarantee mechanism at the business logic layer, we perform multi-tenant CPU resources dynamic evaluation strategy based on a Kalman filter, and the admission control mechanism is used on the basis of the evaluation. The CPU overload threshold is set to 90% according to the experience. Figs 1 and 2 show the CPU utilization and the total CPU utilization of the services S1, S2, S3, S4, separately without an admission control mechanism and with the use of an access control mechanism when the tenants number rapidly fluctuate.

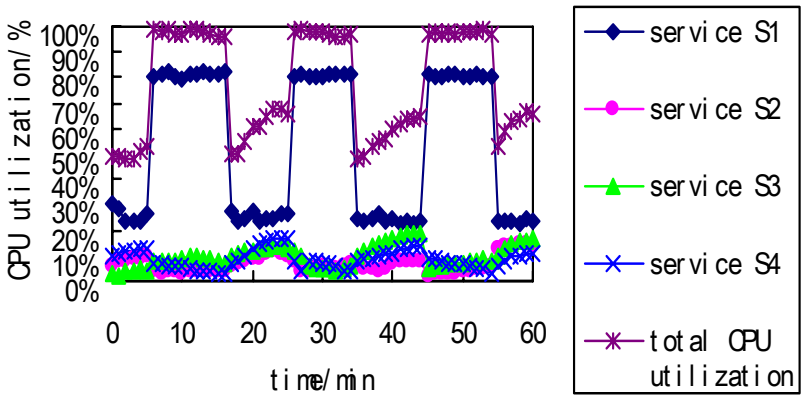

Fig. 1. Without an admission control mechanism

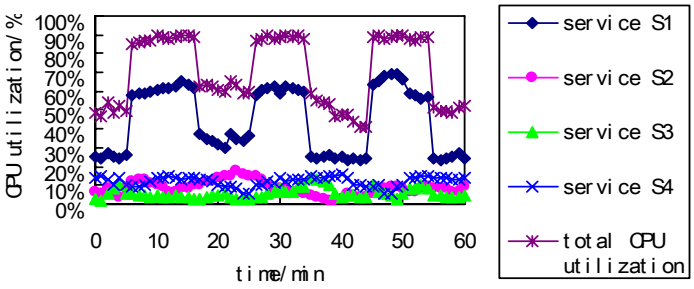

Fig. 2. With an admission control mechanism

As shown in Fig. 1, without the use of an admission control mechanism, with the rapid increase of the number of tenants and the tenant request services S1, S2, S3, S4 at the same time, leading to the server's overload, the server's CPU total utilization rate is close to 100%, which heavily affects the response time. Therefore, the tenants need to wait for a long time. Even some of the tenants' request packet is directly discarded, which seriously affects the user experience. Fig. 2 shows that due to the use of an admission control mechanism, with the same high load, the server's CPU total utilization rate remains below 90%. Although it is in the case of a Flash crowd, it still affects the response time of the request, but the total CPU utilization remains in a reasonable scope, promoting the user experience.

3.2. The data processing layer

In order to evaluate the effect of the performance guarantee mechanism on the data processing layer, we adopt the method of simulation analysis to verify whether GCM can adjust to the dynamic changes of the load through the load distribution and the copy adjustment. We use the widely used and authoritative simulation tool CloudSim.

The parameters of the simulator configuration are as follows: the number of tenants is 1500; the number of copies of each tenant is 4; the number of nodes is 40. In the simulation process, we gradually adjust the overall load intensity to reflect the change of node average resource utilization. We compare the algorithm of OWRR that we proposed, the classical algorithm of Shaped Round-Robin (SRR), Deficit Round Robin (DRR) with the ideal algorithm of the Shortest Queue First (SQF). The comparison results are shown in Fig. 3.

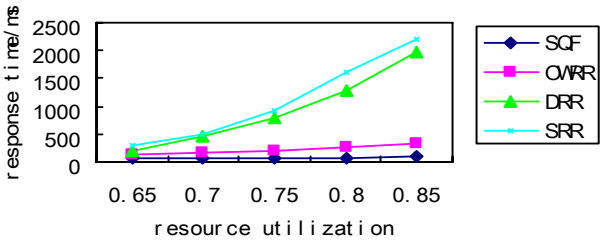

Fig. 3. The performance comparison of four kinds of load distribution algorithms

We can see from Fig. 3 that SQF is an ideal state. With the increase of the resource utilization, there is almost no change in the response time. The performance of OWRR is far better than SRR and DRR, the reason for which is that there is no overall coordination of each tenant load in SRR and DRR, thus leading to a situation where some nodes are under too heavy loads whereas some nodes loads are too light. The performance of processing the tenant requests will be greatly reduced when the load of the node is too heavy. The results show that the OWRR algorithm has a better performance.

It is difficult to deal with the wide margin of fluctuations of the overall load intensity by using the load distribution alone. Then we take the dynamic adjustment of the copy placement into consideration to verify whether GCM can guarantee the performance index of each tenant database on condition that the utilization ratio of system resources is quite high. Taking into account the fact that the system performance will be influenced during the process of copy adjustment, we are looking forward to the minimum times of copy adjustment. Copy adjustment can be refined into creating a copy and deleting a copy. The cost of creating a copy is much higher than deleting one, therefore, in the subsequent experiments, we use the times of creating a copy as one of the main evaluation indices.

Based on the approximate linear relationship between the system load and the response time, we concluded from the experiment that when the tenant number was 1000, the system load reached its limit. Then we tested the situation where the tenant number ranged from 800 to 1500 to determine the efficiency of the safeguard mechanism in the system data processing layer with light load and heavy load. Fig. 4 shows the performance of the algorithm of GCM under the load driver.

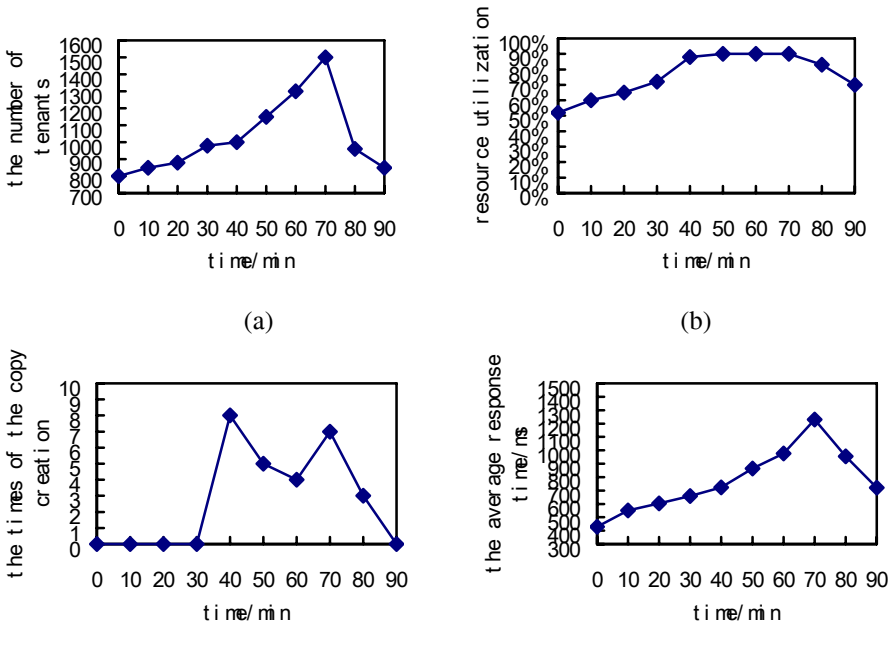

Fig. 4. The performance of the algorithm GCM under the load driver

Fig. 4 (a) shows the curve chart with an independent variable of time when the number of tenants ranges from 800 up to 1500. Fig. 4(b) shows the change of the average resource utilization of nodes as time varies. Fig. 4(c) shows the change of the times of the copy creation. Fig. 4(d) shows the change of the average response time of all tenants as time varies. We can see from the diagrams for the load above given, that GCM can well guarantee the resource utilization on the overall platform and assure that the times of creating a copy are within a reasonable range. A conclusion can be drawn that our safeguard mechanism at the data processing layer has played a good role.

4. Conclusion

The present paper proposed the system architecture SAMSS for multi-tenancy SaaS service, studied SaaS service performance guarantee mechanism respectively at the business logic layer and the data processing layer. For a multi-tenant request processing process at the business logic layer this paper, based on the two aspects of dynamic evaluation of CPU resources and admission control mechanism, puts forward KAC – the Kalman filtering admission control algorithm that has used a Kalman filter to make a dynamic assessment for CPU resources of multi-tenancy SaaS service and response on different server resource usage in time, which would avoid unnecessary performance overhead resulting from an injected probe by direct measurement of the CPU resources. For multi-tenant data processing this paper pertinently studies the two aspects of load allocation and a copy, the optimal weighted rotation scheduling algorithm is used to get the distribution of the load between each copy. This paper brings up the business management approach, the GCM, which loads the balancing issues under the premise of a fixed copy placement formalized as an optimization problem, and puts forward to get the optimal load distribution of the load balancing algorithm. The experimental results show that the GCM can adapt to the dynamic changes of the load through fewer copies of adjustment and guarantee the performance of the tenant database while maintaining high node resource utilization.

Acknowledgements: This work was supported by the National Natural Science Foundation under Grants (No 61472294, No 61171075), the Program for the High-end Talents of Hubei Province, Key Natural Science Foundation of Hubei Province (No. 2014CFA050), and the Open Fund of the State Key Laboratory of Software Development Environment (SKLSDE). Any opinions, findings, and conclusions belong to the authors and do not necessarily reflect the views of the above agencies.

References

1. Mietzner, R., A. Metzger, F. Leymann et al. Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications. – In: Proc. of 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems. IEEE Computer Society, 2009, pp.18-25.
2. Zhang, F., J. Chen, H. Chen et al. Cloudvisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization. – In: Proc. of 23rd ACM Symposium on Operating Systems Principles. ACM, 2011, pp. 203-216.

3. Zhang, Q., L. Cheng, R. Boutaba. Cloud Computing: State-of-the-Art and Research Challenges. – Journal of Internet Services and Applications, 2010, pp. 7-18.
4. Liu, F., B. Li, L. Zhong et al. Flash Crowd in P2P Live Streaming Systems: Fundamental Characteristics and Design Implications. – Parallel and Distributed Systems, IEEE Transactions on, Vol. **23**, 2012, No 7, pp. 1227-1239.
5. García, D. F., J. García, J. Entrialgo et al. A qos Control Mechanism to Provide Service Differentiation and Overload Protection to Internet Scalable Servers. – Services Computing, IEEE Transactions on, Vol. **2**, 2009, No 1, pp. 3-16.
6. Zhang, Yi, Zhihu Wang, Bo Gao et al. An Effective Heuristic for On-Line Tenant Placement Problem in SaaS. 2010 ieee International Conference on. IEEE, 2010, pp. 425-432.
7. Cherkasova, L., K. Ozonat, N. Mi et al. Automated Anomaly Detection and Performance Modeling of Enterprise Applications. ACM Transactions on Computer Systems, Vol. **27**, 2009, No 3, pp. 6.
8. Shvachko, K., H. Kuang, S. Radia et al. The Hadoop Distributed File System. Mass Storage Systems and Technologies. – In: Proc. of 2010 IEEE 26th Symposium on IEEE, 2010, pp. 1-10.
9. Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. – Journal of Basic Engineering, Vol. **82**, 1960, No 1, pp. 35-45.
10. Bovier, A., I. Kurkova. Poisson Convergence in the Restricted k-Partitioning Problem. – Random Structures & Algorithms, Vol. **30**, 2007, No 4, pp. 505-531.
11. Lazowska, E. D., J. Zahorjan, G. S. Graham et al. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Inc., 1984.
12. Bansal, S., S. Sharma, I. Trivedi. A Dynamic Replica Placement Algorithm to Enhance Multiple Failures Capability in Distributed System. – International Journal of Distributed and Parallel Systems (IJDPS), Vol. **27**, 2011, No 3.
13. Yoon, J. H., J. H. Choi, K. J. Lee et al. A Fully Distributed Replica Allocation Scheme for an Opportunistic Network. – Wireless Networks, 2013, pp. 1-13.
14. Li, B., S. Song, I. Bezakova et al. Energy-Aware Replica Selection for Data-Intensive Services in Cloud. Modeling, Analysis & Simulation of Computer and Telecommunication Systems. – In: Proc. of 2012 IEEE 20th International Symposium on. IEEE, 2012, pp. 504-506.
15. Huang, X., Y. Peng. A Novel Replica Placement Strategy for Data Center Network. – In: Proc. of 2012 International Conference on Information Technology and Management Science Proceedings. Berlin Heidelberg, Springer, 2013, pp. 599-609.
16. Tu, M., I. L. Yen. Distributed Replica Placement Algorithms for Correlated Data. – The Journal of Supercomputing, 2013, pp. 1-29.
17. Méndez Muñoz, V., G. Amorós Vicente, F. García Carballeira et al. Emergent Algorithms for Replica Location and Selection in Data Grid. – Future Generation Computer Systems, Vol. **26**,2010, No 7, pp. 934-946.